TP2_Documentação

Equipe:
Arthur Silva Matias - 22052559
Hannah Lisboa Barreto - 22053199

1. Introdução

Este trabalho visa a criação de um sistema de armazenamento e busca para dados de artigos científicos. O sistema é implementado em C++ e utiliza as estruturas de dados de *hashing* e B+Tree para indexação e busca eficientes. O armazenamento dos registros e a criação dos índices seguem as técnicas de organização de arquivos e indexação estudadas nas aulas.

2. Estrutura de Arquivos de Dados e Índices

2.1 Arquivo de Dados

- **Organização**: O arquivo de dados é organizado por *hashing*
- **Campos do Registro**: Cada registro de artigo possui os seguintes campos:
    - **ID** (inteiro): Identificador único do artigo.
    - **Título** (string, até 300 caracteres): Título do artigo.
    - **Ano** (inteiro): Ano de publicação do artigo.
    - **Autores** (string, até 150 caracteres): Lista de autores.
    - **Citações** (inteiro): Número de vezes que o artigo foi citado.
    - **Atualização** (string, data e hora): Data e hora da última atualização do registro.
    - **Snippet** (string, entre 100 e 1024 caracteres): Resumo do artigo.
- **Hashing**: O cálculo do índice na tabela de hashing é feito com a função **hashFunction**, que distribui os registros com base no campo **ID**.

2.2 Índices

O sistema utiliza duas B+Trees como índices:

- **Índice Primário (ID)**: Organizado com base no campo **ID** de cada registro.
- **Índice Secundário (Título)**: Organizado com base no campo **Título** de cada registro, onde o Título é convertido para uma chave hash antes de ser indexado na B+Tree.

3. Estrutura dos Programas

Cada programa é responsável por uma funcionalidade específica.

3.1 upload.cpp

- **Descrição**: Este programa lê os dados de entrada de um arquivo CSV e cria o arquivo de dados organizado por hashing. Em seguida, cria os índices primário e secundário utilizando B+Trees.
- **Funções**:
    - **upload**: Lê o arquivo CSV, carrega os dados na tabela de hashing e insere os registros nas B+Trees.

3.2 findrec.cpp

- **Descrição**: Realiza a busca direta no arquivo de dados por um registro com um **ID** específico, usando a tabela de hashing.
- **Funções**:
    - **hashFunction**: Calcula o índice na tabela de hashing com base no **ID.**
    - **findrec**: Realiza a busca pelo **ID** diretamente na tabela de hashing e exibe os campos do registro encontrado.

3.3 seek1.cpp

- **Descrição**: Busca o registro com **ID**, usando o índice primário (B+Tree). Exibe todos os campos do registro, a quantidade de blocos lidos e o total de blocos do índice.
- **Funções**:
    - **seek1**: Busca o registro pelo **ID** na B+Tree primária, conta os blocos lidos e exibe o resultado.

3.4 seek2.cpp

- **Descrição**: Busca o registro com **Título**, usando o índice secundário (B+Tree). Exibe todos os campos do registro, a quantidade de blocos lidos e o total de blocos do índice.
- **Funções**:
    - **seek2**: Usa o título como chave na B+Tree secundária para localizar o registro e conta os blocos lidos durante a busca.

3.5 BPlusTree.hpp

- **Descrição**: Define e implementa a estrutura de dados B+Tree, usada nos índices primário e secundário.
- **Funções**:
    - **insert**: Insere um registro na árvore.
    - **search**: Busca um registro na árvore e conta os blocos lidos.
    - **splitNode**: Divide um nó da B+Tree quando ele atinge o limite de chaves.
    - **display**: Exibe a estrutura da B+Tree, usada para depuração.

3.6 BlockManager.hpp

- **Descrição:** É responsável pelo gerenciamento dos blocos de dados no sistema. Ele é essencial para garantir que os dados possam ser lidos e escritos em unidades de bloco fixas, o que é comum em sistemas de armazenamento.
- **Funções:**

- **lerBloco:** Lê um bloco do disco.
- **EscreverBLoco:** Escreve um bloco no disco.
- **LerCampo:** Lê um campo de um registro de um bloco.
- **EscreverCampo**: Escreve em um campo de registro.
- **setCatalogo:** inicia o cataĺogo.
- **carregarCatalogo**: Lê o catálogo da memória secundária.
- **atualizarCatalogo:** Salva as alterações feitas no catálogo na memória secundária.

3.7 CSVReader.hpp

- **Descrição:** Fornece as funcionalidades para leitura de arquivos CSV e é usado principalmente no upload.cpp para carregar os dados iniciais.

3.8 DiskManager.hpp

- **Descrição:** Gerencia o acesso ao disco, sendo responsável por operações de leitura e gravação em arquivos persistentes.
  - **memoryAlloc:** aloca memória secundária no disco e devolve seu endereço parecido com malloc do c;
  - **write:** Escreve no disco;
  - **read**: Lê do disco;.
  - **sincronizar:** efetiva as mudanças feitas para o disco;
  - **saveDiskMetaData**: salva as informações de alocação de memória e espaço em um arquivo .dsk;
  - **loadDiskMetaData:** Carrega os metadados do arquivo;

3.9 HashManager.hpp

- **Descrição:** Gerencia o arquivo de hash, além de salvar meta-dados do hash no catálogo.
- **Funções:**
  - **saveHash:** salva o hash table na memória secundária.
  - **loadHash:** carrega o hash da memória secundária.
  - **inserirNoHash:** insere um registro no hash usando o id como campo de hash.
  - **buscarNoHash:** busca um registro no hash utilizando o id.
  - **getQuantidadeDeBlocosLidos:** retorna a quantidade de blocos lidos na última busca.

3.10 OperationSystemDescriptor.hpp

- **Descrição:** Define parâmetros e configurações do sistema operacional, tais como tamanhos de bloco e outras limitações de hardware que o programa precisa respeitar.
- **Funções:**
  - **getDevicesInformation:** Dá as informações dos dispositivos de memória secundária do computador;

4. Papel de Cada Função e Comportamento Geral do Sistema

Cada função foi projetada com um objetivo claro:

- **upload.cpp**:
    - **upload:** Função principal que organiza os dados em hashing e nas B+Trees.
    - **hashFunction:** Distribui os registros na tabela de hashing.
- **findrec.cpp**:
    - **findrec:** Realiza a busca direta no arquivo de dados (tabela de hashing) e exibe o registro.
- **seek1.cpp**:
    - **seek1:** Função que faz a busca pelo índice primário (B+Tree) pelo campo **ID**, exibe o registro e conta os blocos lidos.
- **seek2.cpp**:
    - **seek2:** Função que faz a busca pelo índice secundário (B+Tree) pelo campo **Título**, exibe o registro e conta os blocos lidos.

5. Decisões de Projeto

5.1 Organização do Arquivo de Dados

Optamos por organizar o arquivo de dados com uma tabela de hashing, pois oferece buscas rápidas quando a chave exata (ID) é conhecida.

5.2 Estrutura de Índices com B+Tree

Escolhemos a B+Tree para os índices primário e secundário devido à sua eficiência em operações de busca e sua estrutura balanceada, que mantém o tempo de acesso consistente.

5.3 Organização do Código

A estrutura modular, com cada funcionalidade principal em um arquivo separado, facilita a manutenção, leitura e testes isolados de cada programa.

5.4 Controle de Erros e Manutenção

Implementamos mensagens de erro simples para situações como falha ao abrir o arquivo. Futuros aprimoramentos podem incluir logs detalhados.